DEPARTMENT OF NATIONAL DEFENCE
CANADA

OPERATIONAL RESEARCH DIVISION

DIRECTORATE OF OPERATIONAL RESEARCH (JOINT & LAND)

DOR(J&L) RESEARCH NOTE RN 9716

JANUS RTX IMPROVEMENTS

by

Evan Davies
(Coop Student)

DECEMBER 1997

OTTAWA, CANADA

National Defence Défense nationale

102-02375/

OPERATIONAL RESEARCH DIVISION

CATEGORIES OF PUBLICATION

ORD Reports are the most authoritative and most carefully considered publications of the DGOR scientific community. They normally embody the results of major research activities or are significant works of lasting value or provide a comprehensive view on major defence research initiatives. ORD Reports are approved personally by DGOR, and are subject to peer review.

ORD Project Reports record the analysis and results of studies conducted for specific sponsors. This Category is the main vehicle to report completed research to the sponsors and may also describe a significant milestone in ongoing work. They are approved by DGOR and are subject to peer review. They are released initially to sponsors and may, with sponsor approval, be released to other agencies having an interest in the material.

Directorate Research Notes are issued by directorates. They are intended to outline, develop or document proposals, ideas, analysis or models which do not warrant more formal publication. They may record development work done in support of sponsored projects which could be applied elsewhere in the future. As such they help serve as the corporate scientific memory of the directorates.

ORD Journal Reprints provide readily available copies of articles published with DGOR approval, by OR researchers in learned journals, open technical publications, proceedings, etc.

ORD Contractor Reports document research done under contract of DGOR agencies by industrial concerns, universities, consultants, other government departments or agencies, etc. The scientific content is the responsibility of the originator but has been reviewed by the scientific authority for the contract and approved for release by DGOR.

DEPARTMENT OF NATIONAL DEFENCE

CANADA

OPERATIONAL RESEARCH DIVISION

DIRECTORATE OF OPERATIONAL RESEARCH (JOINT & LAND)

DOR(J&L) RESEARCH NOTE RN9716

JANUS RTX IMPROVEMENTS

by

Evan Davies
(Coop Student)

Recommended by: _____
Maj J.H. Furnivall
A/RWGT

Approved by: _____
G. Lafond
DOR(J&L)

Directorate Research Notes are written to document material which does not warrant or require more formal publication. The contents do not necessarily reflect the view of ORD or the Department of National Defence.

OTTAWA, ONTARIO DECEMBER 1997

ABSTRACT

This report documents the development of the pixmap cache, which speeds up reloading of terrain maps, thus reducing frustrating periods of inaction for the gamers. Several other features of the rtx code encountered during program investigation have also been documented to aid in future code revisions.

TABLE OF CONTENTS

PAGE

ABSTRACT i
TABLE OF CONTENTS ii
GLOSSARY iii
INTRODUCTION 1
AIM 2
INVESTIGATION 2
DESCRIPTION OF SOLUTION 3
RESULTS 3
CAVEAT 4
FUTURE WORK 4
REFERENCES 5
ANNEX A - DISCOVERIES LEADING TO THE DECISION TO USE PIXMAPS A-1
ANNEX B - THE PIXMAP CACHE (view_1_cache) B-1
ANNEX C - AN EXAMINATION OF THE CODE REVISIONS C-1
ANNEX D - THE CODE D-1

GLOSSARY

Janus:	A wargame simulation used by the Canadian military. Originally run on tektronix terminals. The code is written in Fortran
rtx:	The interface between the tektronix terminal and the HP UNIX workstation. The code is written in C and Xlib.
segment:	A structure containing one part of each terrain, such as a road, its location on the screen, and other information.
ted (terrain editor):	The program responsible for creation and editing of terrain maps. The code is written in Fortran.
X:	The name applied to both X Windows and Xlib (the library of functions for X Windows).

JANUS RTX IMPROVEMENTS

INTRODUCTION

1. The change from the VMS/Tektronix Janus version to UNIX Janus greatly increases the speed of drawing the initial map to the screen -- whereas the original system required twenty minutes to load the map, the standard load-time now is less than two minutes. Despite this improvement, a redisplay of the map after a zoom takes longer using UNIX Janus; also, because of the asynchronous nature of the drawing, the gamer loses all input control during a screen refresh. Many maps require more than a minute to complete the redrawing process, during which time the gamer may lose many units to enemy fire -- lost because of his inability to interact with the program while the screen redraws.

2. Until now, a frequent solution to the problem of slow running speeds in Janus has been the use of the smallest possible terrain map, and the removal of any complicated terrain types. Also, restricted use of the zoom features reduces the frequency of map redisplay. The drawing functions in X have become a focus in the investigation, as gamers complain frequently about the waiting period during redisplay of a map -- waiting made necessary by a zoom operation or a window resizing. Gaming would improve a great deal if this period of inaction could be reduced.

3. Zooming and window resizing result in the necessity of map redisplay. When zooming in or out, the user selects the degree of zoom in the menu section, and a bounding box, corresponding to the degree of zoom, appears on the screen. The terrain features within the box will then form the new map when the zoom operation is complete. This feature results in long waiting times between selection of the new map, and reappearance of the cursor after redrawing of the map. Window resizing also requires a redisplay of the terrain, as the selected area of the map must fit the largest possible square area in the window (which changes with any window resizing). Window resizing occurs less frequently than zooming operations, but still has an impact on game speed; because the simulation is played in real time, every second of the game counts.

AIM

4. Several different methods of increasing the operating speed of the simulation have been examined. The first involves an examination of the memory use by the program, while the second studies the control interface between Janus and rtx. The third stores copies of window contents for use in future screen redraws -- an approach ultimately adopted as the best means of increasing speed.

5. Each method, with its area of investigation, will be described in moderate detail, while the Annexes provide an in-depth discussion of the coding changes to the program.

INVESTIGATION

6. Janus, a wargame simulation, runs on UNIX workstations through an interface program called rtx. The interface, rtx, has several important components in separate files: command processing, drawing functions, workstation controlling procedures, memory allocation routines, and structures holding important information. Although the new system of UNIX workstations results in improved performance over the older VAX/tektronix system, further performance improvements remain necessary. This section discusses three possibilities for increased simulation speed, and the following section describes the selected solution.

7. Two structures in rtx, segments and primlists, require large amounts of memory. A segment contains one part of each terrain map: a section of road, river or forest, or possibly a unit. A primlist contains an array of graphics primitive arguments for the X drawing routines. Each segment does not require much memory, but when combined with the primlist structure, the segments used by rtx can occupy as much as 60 MB of memory. Segment and primlist counting code reveals that the program uses that memory efficiently. No simple reworking of the code could result in less memory usage by the program. Note that additional RAM could reduce the amount of paging, thus providing speed-up in some cases.

8. The parse.c file contains coding for the main interface between the Janus program and the rtx interface to the UNIX workstation. The rtx program receives commands from Janus through instruction strings, and these strings (lists of characters) result in different rtx function calls. This method of function call, parsing, can be quite effective, but is generally quite slow. The program operating speed could potentially increase a great deal through use of more

direct function calls, but any change to the present method of parsing would require nearly complete reworking of the rtx program as a whole.

9. Since the main problem with the simulation involves long periods of waiting between zooms, saving a copy of several different maps seems the best approach for increased running speed. X provides a means of saving screens through a pixmap resource, which stores the information of a window, can be drawn to while iconized or invisible (unlike windows), and does not require much memory.

10. The pixmap method provides the best means of significantly increasing Janus operating speed. Annex A provides greater detail on all of the examined methods.

DESCRIPTION OF SOLUTION

11. The new version of rtx saves a copy of the full map in a pixmap. When the user selects the full view of the map, the pixmap copies its contents to the screen; the contents of the pixmap consist of the terrain features -- no unit positions can be included in the pixmap, as the units move during the game. The program stores a new pixmap if the window size changes, as any size change invalidates the previously stored cache. Further coding changes allow for storage of multiple maps, so that the previous and next button features of Janus bring up preselected map views quickly.

12. Previous programmers attempted a similar use of pixmaps, but without success. The changes to rtx use a format similar to that designed in the previous attempt, but otherwise the changes are complete -- the only remaining portion of the original code is the structure that stores the pixmap. The storage of multiple pixmaps has no precedent in the old coding. A full explanation of pixmap implementation can be found in Annex B.

RESULTS

13. The use of pixmaps does not affect the first drawing of the map, so start time remains slow; however, while start time remains unaffected, many advantages of pixmap use make the changes to rtx worthwhile. Pixmaps draw to the screen immediately using an X copy routine, so that the drawing operation lasts less than one second -- a vast improvement from the usual thirty-second-plus redraw times. Window resizing results in the need to redraw the entire

map to the new screen size. However, the pixmap stores the new full map after the first redraw, so a full redraw must occur only once. Pixmap use not only greatly decreases redrawing times, but makes the frequent use of zooms possible.

14. Multiple map caching decreases the time for reloading of zoom-ins dramatically. Use of the previous and next buttons is, however, necessary for this section of coding to be effective -- these buttons form the only means by which the cache and the current zoom level match exactly. Window and cache must match exactly before the cache will copy to the screen. An explanation of all changes to the rtx code can be found in Annex C, and the revised code can be found in Annex D.

CAVEAT

15. The new version of rtx will only work correctly with a version of Janus that has been modified as per Reference 2.

16. When used with ted, it should be invoked with option "-no_cache_view_1".

FUTURE WORK

17. Without eliminating the parse.c method, the number of network messages sent could be greatly reduced. The number of possible commands sent to parse.c sits well under the 255 mark, which is the maximum number that can be set with one byte (256, from zero to 255). If a one byte number identifies a command, instead of the present use of strings, only eight bits need to be sent per command -- far better than the variable length strings used presently. This modification to the coding would affect both Janus and rtx, as the format of commands sent by Janus, and the format of commands accepted by rtx, would have to change.

18. The other change involves the drawing process. Presently, each segment draws to the screen separately, so that the map assembles slowly on the screen. Instead of drawing to the screen each time, perhaps a pixmap created in ted could be stored as a file and loaded into the window when the program opens. The saved file must have the same dimensions as the window, and any window resizing would require a redrawing of the map.

REFERENCES

1. DLOR Research Note RN 9503, "Janus Overview", D.G. Brown, June 1995

2. Note to File, 3552-22242-10, 15 December 1997

ANNEX A
DOR(J&L) RN 9716
DECEMBER 1997

DISCOVERIES LEADING TO THE DECISION TO USE PIXMAPS

Segments and Primlists

1. Ted and Janus use segments in very different fashions. In ted, each section of road, river, or forest is a single segment, which allows the user to modify, for example, one section of road, river, or forest without changing any other sections; each section is one segment, and is, therefore, individually selectable. Janus, on the other hand, holds all sections of road in one segment, all contours in another segment, all rivers in another segment, and so on. Each unit uses one segment, so that one unit can be selected without selecting all units on the map.

2. The Segment structure contains a large number of values used for many different purposes: indicating the position of the segment on the map, the segment's number (or ID), its priority for on-screen display, pointers to both the next segment in the list and graphics primitives for drawing the segment to the screen, and the bounding box containing all parts of the segment. The pointer in the Segment structure, Primlist *prims, allows the segment to reference its first Primlist structure. Primlists require a great deal of memory and their purpose was not immediately clear from the scant program documentation. The following has since been discovered concerning the structure and function of the Primlist: each segment, as mentioned previously, has a pointer to the beginning of a list of Primlist structures, each Primlist has a pointer to the next Primlist in sequence, the *prims pointer, and the maximum size of the primbuf array is set by a preprocessor constant called MaxPrimSize. The overall purpose of the Primlist has also been determined.

3. Primlists contain an array storing integer values; this array, the primbuf, stores information for X graphics routines. Large integers in the primbuf correspond to an x and y screen location, and smaller numbers, Op codes for example, tell the computer what to do at these screen locations. Other members of the Primlist structure include a pointer, and the array size. Groups of primlists are necessary for more complicated segments, so pointers keep these groups in a list. The array size shows how much of the array has been filled with

drawing primitive values. MaxPrimSize can be changed by altering the sizes of its constituent parts: PrimBufSize indicates the maximum size of the primbuf array, and Op_Max_Size leaves room for X operators (such as Op codes, or related information -- font types, colour codes) which are absolutely necessary for X graphics routines. Tinkering with these two values led to some improvement in memory usage by the program, but the improvements were small enough, and unpredictable enough, to discourage any permanent changes either to PrimBufSize or to Op_Max_Size.

4. Changes to the Primlist structure and to its use were ultimately ruled out and this part of the program, in tksim.c, has been left unaltered.

Parse.c and the ct_ Functions

5. An explanation of the mechanics of parse.c follows. Rtx uses parsing to call each function: Janus passes a character string, containing a ct_ function name and one or more numbers, to rtx. The function name is then used to set a function pointer, and the numbers are changed from a character string to float or integer arguments and stored in an array. The function pointer then calls the appropriate ct_ function, and the ct_ function and its subfunctions are processed. The parsing method leads to slower operating speeds, and indirect, often untraceable, function calls. Obviously, changing the method of function calls would be desirable, but this sort of change requires complete restructuring of rtx and Janus, *which is not* easily done. Any changes to this section of the program were also ruled out.

Pixmaps and their Application to rtx

6. Pixmaps have many different purposes in X, including storage for icon bitmaps, cursor bitmaps, window background tiles/stipples, or as copies of window contents in rtx. The application of pixmaps to rtx appeared quite straightforward, as some attempts at pixmap use had already been attempted by American programmers. Pixmaps can be quite versatile, as multiple pixmaps of various map views can be stored in memory simultaneously.

7. The first, unsuccessful, attempt at pixmap application involved the creation of both window and pixmap, with graphics routines drawing to the pixmap and then copying the completed terrain to the window. Drawing routines were expected to run more quickly if graphics were not drawn immediately to the screen, but rather to a pixmap as a temporary

holding area for the screen contents. Attempts at this method on a simplified window program suggested that, contrary to expectations, time savings were negligible, and any savings varied greatly from one trial to the next. Another solution was eventually reached.

ANNEX B
DOR(J&L) RN 9716
DECEMBER 1997

THE PIXMAP CACHE (view_1_cache)

REASONS FOR DEVELOPMENT OF THE view_1_cache

1. Janus draws the full map (or farthest zoom-out) on program commencement. This drawing process can last several minutes, depending on the choice of terrain file -- some are more complicated than others. On subsequent draws of the full map, much of the map information may be stored in memory, as the complete drawing process proceeds much more quickly. However, drawing does not progress quickly enough for the purposes of real-time gaming, so gamers rarely employ the potentially very useful full map. A preferable situation would be one in which the map need not be redrawn with each zoom-out, but rather copied quickly from some existing stored version of the map. In this situation, the reloading of the full map would be almost instantaneous, and a great deal faster than even the highest level zooms -- the full map would suddenly become both practical and useful for the purposes of gaming.

2. By means of a pixmap, a view of the screen can be retained in memory, making any redrawing of this stored view unnecessary. The drawing of the map occurs in several stages, which must be kept in mind for the creation of the pixmap. Drawing order is responsible for the layered appearance of the map: the terrain polygons (urban areas and forests) must appear below the contour lines, and roads should appear over top of the contour intervals. The reverse order results in many invisible contour intervals, as the terrain polygons draw over top of them, and the roads will have lines running over them because of the contours. So, drawing begins with the terrain polygons, then the contour intervals, rivers, roads, and finally obstacles and units. A useful pixmap must contain only the terrain features, as the units and obstacles move during the game -- the map must show the current positions of all movable features of the game, and not be restricted to displaying the positions of these features at startup. If the map cache is to meet the above criteria, the screen contents must be copied to the pixmap before the placement of the units and obstacles. Later, when the pixmap has

been copied to the screen, the current positions of all units and obstacles can be drawn over top of the terrain.

THEORY BEHIND THE CODE

3. A previous attempt at making a cache of the largest view did not work for two major reasons: the code called for pixmap creation at the wrong time, and the values of some arguments were incorrect. As mentioned in the body of the report, some of the overall structure of the coding changes mimics these earlier attempts, and the view_1_cache structure itself has been retained in the current version of rtx as an easy and practical method of cache data storage.

4. A simple approach to pixmap creation would be to wait for all segments to be copied to the screen: when the pointer to the next segment in the list equals NULL (no more segments in the list), the pixmap cache stores the contents of the window -- unfortunately, obstacles and units have already been drawn by this point. Because of the requirement that the pixmap be stored before units and obstacle appear on the screen, an alternative method of cache storage applies. This method exploits the form of function calls by parse.c to the ct.c functions. The appropriate moment for copying the screen to the pixmap occurs when a call to ct_set_segment_visibility() follows a call to ct_select_view() for the first time. The criteria for a successful cache of the full map have already been described in some detail, and now, with the sequence of function calls before pixmap creation clear, the best place for code implementation has to be identified. A discussion of cache implementation follows below.

PIXMAP IMPLEMENTATION

5. The following explanation of programming changes has been kept relatively simple. Changes to the coding centre largely on paint.c and viewcache.c, but minor revisions have also affected parse.c.

Parse.c

6. Parseit() converts a character string, passed from Janus to rtx, into a function pointer and a series of numerical arguments passed to the ct.c functions. These functions lead to the call of all other functions in rtx. The best place for cache assignment occurs when a call to

ct_select_view() precedes a call to ct_set_segment_visibility(), both of which are calls passed to parseit() from Janus. Any pixmap creation routines should, therefore, be called from parseit().

7. The cache will be stored if view one caching is enabled, and if the routines have been called in the expected order. View one corresponds to the largest section of the window, or the playing area itself; the other views involve the menu and smaller, corner map, or specialized parts of the playing area. Only view one should be stored for future zooms. Coding added to parseit() determines whether the routines have been called in the correct order for pixmap creation: a flag is set when ct_select_view() is called; if the next function called is ct_set_segment_visibility(), then a pixmap creation routine assigns the cache and sets a flag to indicate that the cache has been successfully created. However, if the next function called, after ct_select_view(), is not ct_set_segment_visibility(), the flag is unset. This first revision of the code led to problems on some machines, as another check required that a member of the Segment (the views member) equalled 1 -- not a good check, but successful at first. Added precautions were necessary: the program no longer uses the unsuccessful check, and now ensures that the flag is unset if the number accompanying ct_select_view() is not seven.

Paint.c

8. The drawing routines used by the program can be found in the paint.c file. Several of these functions also required a great deal of reworking for the cache to be effective, including several changes to the paint() function, the paint_segment() function, the update_all_view_scale_offs() and update_view_scale_offs() functions, and the paint_view() function.

9. The paint() function redraws the map when a window resizing occurs. The first code revision permitted drawing to a resized window to occur twice before the cache was properly created and stored. Now, however, paint() calls my_paint_segment() instead of paint_segment(), and the cache stores properly on the first drawing of the full map in the resized window. The functions, my_paint_segment() and paint_segment(), have only one major difference between them: my_paint_segment() draws all views to the window except view one (since the cache must be properly created and stored by another function), while paint_segment() draws all views to the screen.

10. The update_all_view_scale_offs() and update_view_scale_offs() functions have been changed only very slightly. The cache was not frequently drawn to the screen because of problems pertaining to floating-point inaccuracies in functions setting values of the views array. The changes to these two functions ensure that the values in the views array will not change, once they have been set, unless the window is resized. These changes, however, affect view_1_cache far less than they affect view_1_multi_cache -- discussed later.

11. The paint_view() function has been changed a great deal. Originally one function, paint_view() has split into two separate functions: paint_view() and paint_from_cache(). Paint_view() still draws all direct segments to the screen -- the obstacles and units -- and draws the view border, but paint_from_cache() is now responsible for the vast majority of all drawing activity. Paint_from_cache() allows for many different drawing situations, including the drawing of views other than view one, storing the cache of the full map (either after the initial drawing, or after window resizing), copying from the cache to the screen, drawing zoom-in maps, creating caches of the zoom-in maps, and copying from zoom-in caches to the screen. The last two subjects will be discussed in the view_multi_cache section.

Viewcache.c

12. Nothing except the view_1_cache structure remains of the original viewcache.c file.

13. The only remaining part of the flag_view_as_changed() function is its name -- the function now determines whether the window size and the size of the stored cache match. If the window size has changed, the window will be redrawn and the contents will be stored to a new view_1_cache pixmap.

14. Create_view_1_cache_pixmap() checks whether a valid cache has already been stored, and then either creates a new cache or returns to its calling function. The function also stores the size of the pixmap, the size of the window, and the pixmap resource ID in the view_1_cache structure.

15. Copy_view_1_cache_to_screen() ensures that the full map has been selected, and that the cache is valid before it copies the pixmap contents to the screen. A return value indicates the successful completion of the copying procedure.

16. Free_view_1_cache() frees the pixmap resource before program termination. KeyPress events must be enabled in rtx.c before this function can be called. KeyPress events should only be used for debugging purposes, so rtx uses free_view_1_cache() very rarely.

THE MULTIPLE PIXMAP CACHE (view_multi_cache)

Reasons for Development of the view_1_multi_cache

17. After the full map has been drawn to the screen and the cursor has appeared, smaller areas of the full map can be selected for magnification. The zoom bar, in the menu section of the window, assists selection of different levels of map magnification. A bounding box appears on the screen when the user selects one of the zoom bar settings -- the area of the map within the box will be enlarged into the view one section of the window (the playing area). The user then watches as the magnified section of the map draws to the screen; although zoom-in views of the map appear on the screen far more quickly than the largest map, users still become frustrated by the waiting period between the view selection and the reappearance of the cursor. Pixmap view caching decreases much of this waiting time, as described below.

Theory Behind the Code

18. The sequence of drawing operations in a magnified map should be the same as in the full map, so an operation of storing several of these magnified views can be made to mimic the view_1_cache creation and storage. Many maps of differing zoom levels should be cached, however, so an array of view_multi_cache structures should be used instead of the single view_1_cache structure. Another difference between the view_1_cache and view_multi_cache structures lies in the requirement to save the size of the map that was cached -- the area zoomed-in on -- so that the correct magnification of the map, at the right coordinates, appears on the screen. A flag indicating correct storage of the cache has also been used. Because of the need to ensure a match between the selected map section and the cache, an array searching algorithm becomes necessary. This search finds a matching array element and moves that element to the top of the array (to the first array position, array subscript zero); this matching cache can now be copied to the screen. If the algorithm does not find a matching array element, the first unused array element moves to the top of the array, and then sets to the new view of the map. All array elements previously above the

selected element shift down one position in the array, with the tenth element being copied over by the ninth if all elements have been used.

19. With these explanations in place, an examination of the method of zooming in Janus may now be useful. The menu section of the window contains two buttons near the zoom bar: Previous and Next. The previous button takes the user back to the last map displayed before the current map. The program retains the ten previous maps, but the current Janus search algorithm is such that only the five previous maps can be redrawn (note that a new version of Janus has eliminated these problems through modification of the next and previous functions). The next button applies only after the previous button has been used, as Next retrieves the map selected before the use of the previous button. The reliance of the multiple cache on these two buttons will be explained in the following section.

PIXMAP IMPLEMENTATION

20. The overall form of the view_multi_cache structure and array has been described above. This section discusses the changes made to several existing functions, and the creation of several new functions for creation and use of the multiple cache. Although parse.c required no changes for the purposes of the multi cache, paint.c and viewcache.c have both suffered many alterations and revamps.

Paint.c

21. The changes made to update_all_view_scale_offs() and to update_view_scale_offs() benefit the multi cache far more than the view one cache. These function alterations prevent changes to the playing area of the window once it has been set; before this alteration, inaccuracies in certain routines allowed the playing area size to change by a pixel, which prevented the proper matching of cache and Views array members. The importance of these changes will be discussed further under the viewcache.c section.

22. The other main changes to paint.c affect the paint_from_cache() function. Paint_from_cache() has several alternatives concerning the multi_cache. The function first checks whether the selected zoom matches any of the previous ten zooms (stored in the multi_cache array). If the search finds no match, the function draws the view to the screen and then copies the screen to the first multi_cache array member. If, however, a matching

cache exists, this array member moves to the top of the array and another function copies it to the screen.

23. Also note that no multi_caches can be stored until the first view_1_cache has been created and stored properly. A single variable controls the entire caching system: if the cache_view_1 variable is set to zero, either in rtx.c or by the command-line arguments, neither view_1_cache nor view_multi_cache coding will be used.

Viewcache.c

24. The first function associated with the pixmap multi cache, check_multi_cache(), determines whether any existing caches match the current map magnification and view coordinates. If a match is found, check_multi_cache() sets a pointer to the location of the matching cache member -- this member moves to the top of the array. If the search finds no match, a pointer marks the first unused cache entry. This entry moves to the top of the array, and the screen draws and then stores the map in this top member.

25. The function responsible for resetting the order of the multi_cache array entries, reset_cache_order(), sets a temporary structure to the contents of the matching cache (or new cache to be created), and then shifts each cache member down one position in the array. If all array members have been filled by a cache, the tenth member moves to the first position, where it will be reset to the current map in the playing area -- all other view_multi_cache members still shift down one position. If the matching cache sits in the middle of the array, with caches set above and below its position, only the caches above its position in the array shift down one place. The matching array then moves to the top of the array, to the first array subscript.

26. When paint_from_cache() calls create_multi_cache_member() for the first time, it creates all the multi_cache pixmaps and sets the only the top pixmap to the screen contents. The create_multi_cache_member() function sets, by default, the top array member; this default to the top cache member explains the importance of the array reordering performed by the reset_cache_order() function. While the window size remains unchanged, no new pixmaps will be allocated. The first time a zoom-in occurs in a resized window, the old pixmaps will be freed and new ones created.

27. The copy_multi_cache_to_screen() function copies the top array member to the screen. The function's only check ensures that the cache has been set properly.

28. The multi_cache_initialize() function sets all values in each of the array members to zero. This function has little purpose, but has been left in the code to ensure proper initialization of the array.

29. The final two functions, free_multi_cache_pixmap() and free_all_multi_cache_pixmaps(), free the pixmaps before exiting the program. The KeyPress case in rtx.c has been disabled in the non-debugging version of the code, so the program no longer uses these functions. They have been left in the code for future debugging purposes.

ANNEX C
DOR(J&L) RN 9716
DECEMBER 1997

AN EXAMINATION OF THE CODE REVISIONS

1. The following section describes the changes to the coding itself. Each file altered by the coding changes will be dealt with in a separate section, and each changed function will be described in detail under the appropriate file heading. Some optional changes will also be discussed, such as the Segment and Primlist counting routines added to tksim.c.

Parse.c

2. All changes made to parse.c apply to the parseit() function. A flag, flag_correct_order, indicates when the view_1_cache must be assigned. When parse() calls parseit() for the first time, the flag initializes to zero (False). When Janus sends the string "ct_select_view..." for the first time, the flag sets to one (True). The next time parseit() runs, the program checks whether the command sent from Janus contains the string "ct_set_segment_visibility," and if so, parseit() calls create_view_1_cache_pixmap(). When the create_view_1_cache_pixmap() function terminates and control returns to the parseit() function, the flag resets to zero. If, however, the command after "ct_select_view" is not "ct_set_segment_visibility," the flag resets to zero. Another important addition to the code can be found toward the bottom of the parseit() function: when the flag has been set True, the function checks whether the number sent with "ct_select_view" is seven. The flag resets to False if the number is not seven, preventing the call of the create_view_1_cache_pixmap() function -- view seven follows view one. Note that the correct argument to verify with "ct_select_view" is view seven and not view one, since "ct_select_view" selects the next view, not the current view.

PAINT.C

Paint()

3. Paint() redraws the screen when the window resizes, and calls several other functions to ensure all values and variables are set properly in the resized view. This function has three changes, not including changes to the functions it calls.

4. The first change involves the addition of a flag, view_X_coords_set. This flag allows for only one setting of the X coordinate values for view one to occur. The flag sets to False initially, and sets to True after the calls to update_view_scale_offs() from update_all_view_scale_offs(). Only view one is affected by this change. Trans_x() and trans_y() reset the views array variables every time they run, but the values set to X_x1, X_y1, X_x2, and X_y2 generally change in value by plus or minus one (floating-point inaccuracies), resulting in improper matches in the view_multi_cache functions. See the update_all_view_scale_offs() and the update_view_scale_offs() sections for further discussion.

5. The second change results in a call to paint_view(), with the viewnum variable set to one. The view_1_cache pixmap will now be set to the proper size on the first redraw of the screen after window resizing, instead of requiring a second redraw. A second redraw occurred initially because the cache was not set in the first call to paint_view() after redrawing of the screen by paint().

6. The final change results in a call to my_paint_segment() instead of paint_segment(). The paint_segment() function could not be changed, as many other functions also require it at different times, so my_paint_segment() takes its place in paint().

Update_all_view_scale_offs() and update_view_scale_offs()

7. The change made to update_all_view_scale_offs() involves the setting of a single flag; this function sets all view array values after a window resize. Setting the view_X_coords_set flag to True affects the update_view_scale_offs() function as follows. Update_view_scale_offs() calls two functions, trans_x() and trans_y(), both of which reset many values in the views array. Once the flag sets to True, along with other requirements

detailed below, these values will not be reset (and risk possible change by plus or minus one) when update_view_scale_offs() uses the values returned by trans_x() and trans_y(). The following statement models the Boolean logic used in the code: the views array values (X_x1, X_y1, X_x2, and X_y2) will not be reset if the view number is one, a valid view_1_cache has been stored or a matching multi_cache has been found, and the view coordinates have already been set for this window size.

My_paint_segment()

8. My_paint_segment() differs from paint_segment in only one respect; it calls the paint_segment_in_view() function for view numbers greater than one.

Paint_view()

9. Only two changes have been introduced to this function. The first change involves the definitions of the maximum size of two variables in the views array, and the second change calls the paint_from_cache() function. The definitions relate to the size of the large background map, of dimension 307200 by 307200, created by Janus. These coordinates must be transformed to X window coordinates for use by rtx. More information on the large map follows. These definitions could be moved to the paint_from_cache function().

Paint_from_cache()

10. Documentation for this function can be found in the coding itself, but an in depth explanation of the function may prove helpful.

11. The initialization of the mask variable remains from the paint_view() function, and serves the same purpose here as in paint_view(). The first change to the coding involves the if statement following variable initialization. If the view number equals one and the view one cache has been stored, the program moves to the next, inner, if statement. Cache_stored sets to one (True) when parseit() calls create_view_1_cache_pixmap(), so the expression in the first, outer, if statement will evaluate to False only before the screen has been drawn to for the first time -- after all, a nonexistent pixmap cannot be copied to the screen. Note, also, that the else statement, at the end of the function and corresponding to the first if, works for all views other than view one.

12. The second, inner, if statement checks whether the selected area in the large, background map is the whole map: the starting coordinates, window_x1 and window_y1, should equal zero, and the end coordinates, window_x2 and window_y2, should equal 307200. The flag_view_as_changed() function, in viewcache.c, then checks whether the window has been resized. If the stored cache remains valid -- window size has not changed -- the function calls copy_view_1_cache_to_screen(). Otherwise the window has been resized, so all segments of priority three and less draw to the screen, and the view one cache resets to this new window size. Paint_from_cache() then draws the remaining features, such as units and obstacles, to the screen and the function terminates.

13. The else, corresponding to the second, inner, if statement, executes when the present map settings do not indicate selection of the full map. The check_multi_cache() function determines whether the present map settings match any of the stored multi_cache array members. If a matching cache exists, the matched_multi_cache flag sets to True, and copy_multi_cache_to_screen() draws the pixmap to the screen. If the flag sets to False, graphics routines draw the map to the screen, and the top array member stores the new map. Paint_from_cache() draws the rest of the features of priority greater than three, such as units and obstacles.

Viewcache.c

14. The global variables will be explained before discussion of each function: view_1_cache, cache_view_1, cache_stored, MaxCache, view_multi_cache, and current_cache. View_1_cache holds five variables, with two variables holding the height and width of the window when the cache was created, two variables holding the pixmap size, and the last variable holding the pixmap resource ID. The flag, cache_view_1, activates screen caching. Cache_stored indicates successful creation and storage of the view_1_cache. MaxCache sets the maximum number of pixmaps to be stored in the view_multi_cache array. View_multi_cache has five members in common with the view_1_cache structure, but also includes five other members. Two of the new members store the cache origin relative to the large background map, two other members store the opposite point on the background map (these four members have values of between 0 and 307200), and the fifth member indicates proper creation and storage of each cache. Creation of the view_1_multi_cache array follows the declaration of the view_multi_cache structure. The view_1_multi_cache array holds

MaxCache (currently ten) members. Finally, current_cache points to a member of the view_1_multi_cache array, and figures prominently in several view_multi_cache functions.

Flag_view_as_changed()

15. This function checks for a window resizing, setting the cache valid flag, cache_stored, to False if the window size has changed. Flag_view_as_changed() affects only the view one cache, not the multi_cache. The function works as follows. The view number must be one, or the function returns to its calling function; this line of coding seems pointless, as the view number must be one even to enter the if statements in paint_from_cache(). Although this check could be safely removed from the function (it is a remnant of the old version of this function), it does no harm, and may as well remain in the program. The large if statement checks map dimensions to ensure that the view consists of the furthest possible zoom-out, and then checks whether the window size has changed. If the window size has not changed, the function returns to paint_from_cache() and the cache remains valid; however, if the statement evaluates to True, the stored cache is no longer valid, as the window size has changed: cache_stored sets to zero (False). The function terminates and control returns to paint_from_cache().

Create_view_1_cache_pixmap()

16. The name of the function indicates clearly its purpose. If the cache has not been created and stored, or the window size has changed, the function checks whether a pixmap resource has been assigned for the cache. If a resource exists, the function frees it, as the old resource is no longer valid. The structure erases all old member contents by replacing these removed values with current values: the new pixmap size and window size. A new pixmap resource set, the function copies the current screen contents to the pixmap, and the cache_stored flag sets to True. When the cache is valid upon function entry, the else statement executes, returning control to its calling function.

Copy_view_1_cache_to_screen()

17. This function, too, works as its name implies. The function first checks that the pixmap has been set, and that caching is enabled. The rest of the checks ensure that the full map has been selected, and that no window resizing has occurred. These checks seem

redundant, since the create_view_1_cache_pixmap() and flag_view_as_changed() check and set all these variables, but they do no harm either. When the if statement evaluates to true, the pixmap appears on the screen and a return value indicates proper function completion. If the if statement evaluates false, a return value indicates improper completion to the calling function. Neither of these return values, however, sees use by paint_from_cache() -- this situation could be rectified, but any changes are unnecessary.

Free_view_1_cache()

18. This function calls XFreePixmap() and returns True if the function is successful. If a pixmap has not been assigned, the function returns False.

Check_multi_cache()

19. The previous functions in this section affect only the view_1_cache; the following functions affect the multi_cache. The check_multi_cache() function checks all multi cache array members for a match between the current zoom values and the stored caches. A for loop cycles through all members of the view_1_multi_cache array, checking for matching pixmap and window dimensions, for matching background map origin and end points, and for assigned pixmaps. The current_cache pointer stores the location of the matching cache, or if no matching cache can be found, the first free array member (or the last member if the array is full) shifts to the top of the array and resets to the current map view. The array reconfiguration and resetting are not part of this function, but result from calls to other functions. A return value of one (True) results in a call to copy_multi_cache_to_screen(), whereas a return value of zero (False) causes paint_from_cache() to draw this new map to the screen and to store the map into the top array position. Note that any change to the MaxCache size will cause errors in this function, so this function must fit the current MaxCache value.

Reset_cache_order()

20. Any matching cache moves to the top of the array, array subscript zero, by means of the decrementing for loop in this function. A temporary structure stores the matching cache member so that it cannot be overwritten during the shift. The current_cache pointer has been set to the correct array member by check_multi_cache(). Notice that this function can be

used either for shifting the matching array member to the top of the array, or to prepare the array for a call to create_multi_cache_member().

Create_multi_cache_member()

21. This is another function that does what its name implies. Create_multi_cache_member() allocates one pixmap resource to each member of the multi cache array on the first call to the function from paint_from_cache(). On subsequent calls, XFreePixmap() frees all assigned pixmaps if the window size has changed -- the purpose of the first if statement in the function. If no window resizing has occurred, the function copies the current map to the top array member and sets all variables in the first array member to current map values. Recall that reset_multi_cache() sorts the array so that the first unused array member (or the last member if all members have been assigned a valid pixmap) receives the current map view. With the first call to the function, or a window resizing, the current map view copies to the first array member. The matched_size flag does not serve any useful purpose and could be removed from the code, but it does clarify certain areas of the function. The pixmap_assigned variable in the view_multi_cache structure serves a very important purpose, as the function assigns all pixmaps on the first time through -- the flag serves to identify pixmaps containing a valid map.

Copy_multi_cache_to_screen()

22. As long as the pixmap_assigned flag is True, the function calls XCopyArea() to copy the pixmap contents to the screen.

Multi_cache_initialize()

23. The for loop ensures that all values in each array member start at zero. The most important line sets the pixmap_assigned flag to False in each multi cache member.

Free_multi_cache_pixmap() and free_all_multi_cache_pixmaps()

24. These two functions are analogous to free_view_1_cache(). Free_all_multi_cache_pixmaps() calls free_multi_cache_pixmap() MaxCache times, freeing all pixmap resources used by the program. Free_multi_cache_pixmap() receives several calls

from other viewcache.c functions, but its only purpose is to call XFreePixmap(). Free_all_multi_cache_pixmaps() receives a call only from rtx.c, and only when debugging is enabled. One concern is that X may not free the pixmaps used by rtx on program termination, but X generally removes unused resources, so the pixmaps are likely removed from memory when rtx terminates. X does not grow beyond a certain size, so the problem of freeing pixmaps is not a large one. One possible solution involves the addition of a ct_quit() function to ct.c.

Tksim.c

25. The details of the segment and primlist counting routines have been omitted. The only important note is that the functions found in the revised version of tksim.c work, and that the use of pointers in the functions does not affect the operation of rtx.

26. To use these functions, simply reset the comment signs (/*) to surround the comments and not the code in the end_segment() function, and define old (for example, #define old 1) at the beginning of the segment_reports() function. The changes to end_segment() will result in a great deal of information printing to the output file as the program progresses; the changes to segment_reports() result in information being printed to the output file when the program terminates.

THE CODE

Changes to Parse.c

```
/* a couple of global variables I had to add */

/* E Davies 97/10/31 -- from rtx.c */
extern int cache_view_1;
extern int cache_stored;


/* input:  line of text from janus graphics
  output: doit, args[], argcount
*/
parseit()
{
        char *t, *index();
        char *s, *p;
        int ix;
        char *fun_name;
        void lookup();
        int rc;

        /* E Davies 97/10/28 -- flag for use in coding below */
        static int flag_correct_order = 0;

        argcount = 0;

        t = index(linebuf, '\n');       /* zap ending nl */
        if (t != NULL) *t = 0;
        else {
```

```
            printf("Warning! line doesn't end in n!\n");
        }

        fun_name = linebuf;          /* line starts w/function name */
        t = index(linebuf, ',');     /* won't have one if no args */
        if (t != NULL) *t++ = 0;     /* zap comma if have one */

/* E Davies 97/10/28 -- find when ct_select_view() is followed by
 * ct_set_segment_visibility() for view_1_cache creation */

      if (flag_correct_order && cache_view_1) {
        if ((strcmp(fun_name, "ct_set_segment_visibility") == 0)) {
          printf("The flagging routine worked...\n");
          create_view_1_cache_pixmap();
          printf("cache_stored = %d\n", cache_stored);
          flag_correct_order = 0;
        }
        else
          flag_correct_order = 0;
      }
      else {
        if (strcmp(fun_name, "ct_select_view") == 0)
          flag_correct_order = 1;
      }
/* end of modification -- E Davies */

        lookup(fun_name);            /* set doit function */

        for (ix = 0 ; ix < MaxArgs ; ix++ ) {
            if (t == NULL) return;       /* if no more args, all done */
            s = t;                       /* set start pointer */
            t = index(s, ',');           /* end this arg */
            if (t != NULL) *t++ = 0;     /* zap comma if have one */
            p = index(s, '.');           /* see if any periods (float) */
            if (p == NULL) {             /* integer arg */
```

```
                int *ip;
                ip = &int_args[ix];
                rc = sscanf(s, "%d", ip);       /* atoi */
                args[ix] = (u_char *)ip;
                if(flag_correct_order && int_args[ix] != 7)  {
                   flag_correct_order = 0;
                }
            }
            else {                      /* float arg */
                float *ip;
                ip = &float_args[ix];
                rc = sscanf(s, "%f", ip);       /* parse as float */
                args[ix] = (u_char *)ip;
            }
            argcount++;
            if (rc != 1) punt("logic err,#1");
        }
        punt("MaxArgs too small");
}
```

Changes to Paint.c

```
/* some global variables I need to use */

/* E Davies -- 97/10/31 */
extern int cache_stored;    /* from rtx.c... set in viewcache.c */
Flag matched_multi_cache;   /* whether to make new cache or use existing one */
void paint_from_cache();    /*  function declaration */

/* E Davies -- 97/11/12 */
Flag view_X_coords_set = FALSE; /* X coords for each view not set */

/* Redisplay all data; both in normal segments and direct-view */
paint()
{
        Segment *segment;
```

```
    int i;

  view_X_coords_set = FALSE;     /* don't redo v->X_.. once set */
    update_all_view_scale_offs(); /* update the view scales */
    push_all_cursors();          /* hide the cursor if on the screen */
    clear_views();               /* erase & box the viewports */

    paint_all_direct = TRUE;      /* flag to paint everything */

    paint_view(1); /* paint view 1, set cache -- E Davies */

  for (segment = seg_pri_list;
       segment != NULL;
         segment = segment->next_pri) {

     my_paint_segment(segment, FALSE); /* 97/11/04 -- E Davies */
  }

    for (i = 1 ; i <= MaxView ; i++ ) {     /* paint all direct data */
          if (views[i] != NULL &&          /* menu highlights etc */
             views[i]->direct != NULL &&
             views[i]->direct->prims != NULL)
           paint_segment_in_view(views[i]->direct, i, FALSE);
    }

    paint_all_direct = FALSE;

    pop_all_cursors();

    XSync(s->display, False);     /****/
}

update_all_view_scale_offs()
{
    int i;
```

```
        for (i = 1 ; i <= MaxView ; i++ ) {
                if (views[i] != NULL)
                        update_view_scale_offs(views[i]);
        }
    view_X_coords_set = TRUE;  /* coords for all views set -- E Davies */

        /* convert pick aperture to X coord too */
        /* pick aperture defines a box centered on cursor, we want a simple */
        /* delta for distance comparisons, use 1/2 of the width of the 'box' */

        /*  let's make it bigger, don't divide by 2 anymore */

        pick_aperture_x = ((double)pick_aperture_tek) *
                        ((double)(s->width) / 4096.) + .5;      /* 4096 was 2*4096 */
}

update_view_scale_offs(v)               /* compute the view to X transforms */
View *v;
{
        trans_x( &v->scale_x,           /* outputs are scales and offsets */
                &v->off_x,
                v->window_x1, v->window_x2, v->viewport_x1, v->viewport_x2,
                s->width);
        trans_y( &v->scale_y, &v->off_y,
                v->window_y1, v->window_y2, v->viewport_y1, v->viewport_y2,
                s->height);

        /* if(viewnum = 1, view 1 coords set properly), don't reset X coords */
        if((v->viewnum == 1) && (cache_stored || matched_multi_cache)
          && view_X_coords_set)         return;

        else {
          v->X_x1 = (v->window_x1 * v->scale_x) +  v->off_x;
          v->X_y1 = (v->window_y1 * v->scale_y) +  v->off_y;
          v->X_x2 = (v->window_x2 * v->scale_x) +  v->off_x;
```

```
            v->X_y2 = (v->window_y2 * v->scale_y) +  v->off_y;
        }

}


/* set view_1_cache with first repaint.  Called by paint() only -- E Davies*/
my_paint_segment(seg, erase)
Segment *seg;
Flag    erase;          /* if true, use view's wipe color */
{
        u_int views;
        u_int viewnum;

        viewnum = 2;
        views = seg->views;
        views = views >> 1;   /* start with bit for view 2 */

        while (views) {  /* cycle through all different views seg appears in */
         if (views & 0x01) paint_segment_in_view(seg, viewnum, erase);
         views = views >> 1;
         viewnum++;
        }
}

/* (re)paint a view, clear out direct paint commands, use/store cache */
/* E Davies -- 97/11/04 */
paint_view(viewnum)
u_int viewnum;
{
        View    *v;
        Segment *segment;

        v = views[viewnum];
        if (v == NULL) punt("oh no");
```

```
        if (v->direct != NULL &&
           v->direct->prims != NULL) {
                del_prims(v->direct->prims);
                v->direct->prims = NULL;
                v->direct->cur_prims = NULL;
                v->direct->direct_prims = NULL;
                v->direct->direct_off = 0;     }

/* E Davies -- 97/11/04 -- cache use and creation... */
#define MaxX     307200   /* cache view 1 if window is this big */
#define MaxY     307200   /*    with origin at 0,0              */

        paint_view_border(v);
        paint_from_cache(segment, viewnum);
}


/* called by paint_view(). Copy cache with zoom out, create cache if needed */
/* E Davies -- 97/11/13 */
void
paint_from_cache(seg, viewnum)
Segment *seg;
int viewnum;
{
        View *v;
        int mask = 1 << (viewnum - 1);

        v = views[viewnum];

        if (viewnum == 1 && cache_stored)  {
           if (v->window_x1 == 0 && v->window_y1 == 0   /* ensure max zoom out */
              && v->window_x2 == MaxX && v->window_y2 == MaxY )  {

                flag_view_as_changed(v->viewnum);  /* check if window size changed */
```

```
  if(cache_stored) {      /* window size hasn't changed */
     copy_view_1_cache_to_screen(); /* then draw rest of non-
                                terrain features */

  }

else    { /* window size has changed */
    for ( seg = seg_pri_list; seg != NULL ;
            seg = seg->next_pri )

          if (seg->views & mask && seg->priority <= 3) {
                paint_segment_in_view(seg, viewnum, FALSE);
             }

     create_view_1_cache_pixmap(); /* copy terrain features */
 }

    /* draw rest of non-terrain features to map */
    for ( seg = seg_pri_list; seg != NULL ;
            seg = seg->next_pri )

        if (seg->views & mask && seg->priority > 3) {
                paint_segment_in_view(seg, viewnum, FALSE);
         }
}

else  { /* not the largest zoom out */

     /* 1 if match, 0 if make new pixmap */
     matched_multi_cache = check_multi_cache();

     if(matched_multi_cache)          /* found matching cache */
       copy_multi_cache_to_screen(); /* then copy all direct
                                  segs to screen */

      else  { /* no matching cache */
```

```
            for ( seg = seg_pri_list; seg != NULL ;
                  seg = seg->next_pri )

              if (seg->views & mask && seg->priority <= 3) {
                    paint_segment_in_view(seg, viewnum, FALSE);
                }

              create_multi_cache_member();
            }

            /* draw rest of non-terrain features */
           for (seg = seg_pri_list ;
                    seg != NULL ;
                      seg = seg->next_pri )

              if (seg->views & mask && seg->priority > 3) {
                 paint_segment_in_view(seg, viewnum, FALSE);
              }

       }
      }

      else {   /* not view 1 or cache not stored */
         for (seg = seg_pri_list ;
                seg != NULL ;
                  seg = seg->next_pri )

            if (seg->views & mask) {
               paint_segment_in_view(seg, viewnum, FALSE);
            }
       }
}
```

Changes to Viewcache.c

```
/* $Id: viewcache.c, 1997/11/13 13:10 -- E Davies */

#include <stdio.h>
#include <sys/types.h>

#include "tksim.h"
#include "rtx.h"
#include "matrix.h"
#include "viewcache.h"

struct {
        int     x_width;        /* width/height of X window when  */
        int     x_height;       /*  the cache was saved           */
        int     pix_width;      /* w/h of pixmap for view 1 cache */
        int     pix_height;
        Pixmap  pixels;         /* pixmap handle for view 1 cache */
} view_1_cache;

#define View_1_MaxX     307200  /* cache view 1 if window is this big */
#define View_1_MaxY     307200  /*  with origin at 0,0                */

extern int cache_view_1;        /* from rtx.c -- using view_1_cache */
extern int cache_stored;        /* from rtx.c -- whether cache is stored */

/* max number of members in multi_cache array */
#define MaxCache    10

struct view_multi_cache
{
   int pix_width;           /* width and height of pixmap */
   int pix_height;

   int x_width;             /* width and height of window when pixmap stored */
   int x_height;
```

```
    int back_origin_x;      /* origin of background map (0->307200) */
    int back_origin_y;

    int back_corner_x;      /* location of opposite corner of background map */
    int back_corner_y;      /*     (0->307200) */

    int pixmap_assigned;  /* 0=pixels not set, 1=pixels set */
    Pixmap pixels;          /* the pixmap */
} view_1_multi_cache[MaxCache];

struct view_multi_cache  *current_cache;  /* currently selected struct member */


extern struct Server *s;        /* from rtx.c */
XVisualInfo *visualList;        /* from rtx.c: for view_1_cache depth... */


/* Following functions deal with view_1_cache, a pixmap copy of the main map */
void
flag_view_as_changed(viewnum)  /* reset cache valid flag */
int viewnum;
{
        View *v;

        v = views[viewnum];

        if (v->viewnum != 1) return;

        if (v == NULL) punt("logic error in flag_view_as_changed");

        if ( (v->window_y1 == 0) &&     /* if full-size zoom ("1" zoom) */
             (v->window_x1 == 0) &&     /* but window size changed */
             (v->window_y2 == View_1_MaxY) &&
             (v->window_x2 == View_1_MaxX) /* &&
             (view_1_cache.pix_width != v->X_x2) &&
```

```
            (view_1_cache.pix_height != v->X_y1) */ &&
            (view_1_cache.x_width != s->width) &&
            (view_1_cache.x_height != s->height)
           ) {
                cache_stored = 0;

            /* printf("flagged view as changed...\n"); */

        }
}

/* E Davies -- 97/10/31  */
void
create_view_1_cache_pixmap()
{
        if(!cache_stored) {
          int view_1_cache_depth = visualList[0].depth; /* set depth */

          if(view_1_cache.pixels)  { /* if pixels already exists, free it */
             XFreePixmap(s->display, view_1_cache.pixels);
             printf("freed pixmap, make another...\n");
          }

          view_1_cache.pix_width = views[1]->X_x2;     /* set width and */
          view_1_cache.pix_height = views[1]->X_y1;    /* height of cache */

          /* probably completely unnecessary, but I'll check anyhow */
          if (view_1_cache_depth == 0)  view_1_cache_depth = 8;
          if (view_1_cache_depth != 8)  printf("cache_depth != 8 !?!\n");

          view_1_cache.pixels = XCreatePixmap(s->display, s->win,
                                view_1_cache.pix_width,
                                view_1_cache.pix_height,
                                view_1_cache_depth);
```

```
            printf("pixmap created.  ID = %d\n", view_1_cache.pixels);

            /* should NEVER happen... */
            if(!(view_1_cache.pixels))  punt("pixmap not created...\n");

            view_1_cache.x_width = s->width;  /* window size when cache created */
            view_1_cache.x_height = s->height;

            /* store contents of screen into created pixmap */
            XCopyArea(s->display, s->win, view_1_cache.pixels,
                  s->gc, 0, 0, view_1_cache.pix_width,
                    view_1_cache.pix_height, 0, 0);

            cache_stored = 1;  /* cache has now been created properly */
          }
          else
            printf("cache already stored\n");
}

int copy_view_1_cache_to_screen()
{
    View *v;

    v = views[1];
    if (v == NULL)  punt("logic error in copy_view_1_cache_to_screen()\n");

    if ((view_1_cache.pixels) && (cache_view_1) && /* if full-size zoom ("1" zoom) */
       (v->window_y1 == 0) && (v->window_x1 == 0) &&
       (v->window_y2 == View_1_MaxY) &&
       (v->window_x2 == View_1_MaxX) &&
       (view_1_cache.x_width == s->width) &&
         (view_1_cache.x_height == s->height))  {

          XCopyArea(s->display,
                view_1_cache.pixels,
```

```
                    s->win,
                    s->gc,          /* still set from above */
                    0, 0,
                    view_1_cache.pix_width,
                    view_1_cache.pix_height,
                    0, 0);
            return 1;
    }
    else
        return 0;
}

/* frees the cache before exiting the program (from KeyPress case in rtx.c) */
int free_view_1_cache()
{
        if(view_1_cache.pixels) {
            XFreePixmap(s->display, view_1_cache.pixels);
            return 1;
        }

        else
            return 0;
}

/* the following code makes several caches of various levels of zoom */
/* E Davies -- 97/11/13 */
int
check_multi_cache()
{
   int i;

   View *v;
   v = views[1]; /* for the test below... */

   if(current_cache)  current_cache = NULL; /* set to NULL before start */
```

```
/* cycle through each struct member to find matching cache */
for (i = 0; i < MaxCache; i++)  {

   if((view_1_multi_cache[i].pix_width == v->X_x2)  && /* pixmap w/h */
     (view_1_multi_cache[i].pix_height == v->X_y1) && /* matches view w/h */
     (view_1_multi_cache[i].x_width == s->width)   &&  /* w/h of win when pixmap */
     (view_1_multi_cache[i].x_height == s->height) &&  /* stored = current size */
     (view_1_multi_cache[i].back_origin_x == v->window_x1) &&
     (view_1_multi_cache[i].back_origin_y == v->window_y1) &&
     (view_1_multi_cache[i].back_corner_x == v->window_x2) &&
     (view_1_multi_cache[i].back_corner_y == v->window_y2) &&
     (view_1_multi_cache[i].pixmap_assigned == True))
   {
      current_cache = &view_1_multi_cache[i];
   } /* end if */

   if(current_cache)  { /* if it's set to point to something */
      reset_cache_order(i);
      return 1;   /* ready to copy pixels to screen */
   }
}

/* no matching cache?  Rearrange order, make new one  */
if(!(current_cache)) {     /* current_cache == NULL ie. no match, */
   /* if all caches are assigned, jump straight to creation of new one,
    * drop number 10 */
   if(view_1_multi_cache[9].pixmap_assigned == True)
        i = 9;
   else { /* all caches are not assigned, look for first unassigned */
      for(i = 0; i < MaxCache; i++)  {
         if(view_1_multi_cache[i].pixmap_assigned == False)  {
            break;
         }
      }
```

```
        }
            current_cache = &view_1_multi_cache[i];
            printf("no matching multi_cache, reordering to %d\n", i);
            reset_cache_order(i);  /* move each entry down, make room for new cache */
            return 0;   /* must make new pixmap in paint_from_cache() */
    }

}

/* move matching cache to the top of the list, shift all others down one */
void
reset_cache_order(matched_cache)
int matched_cache;   /* position of matching structure member in array */
{
    int i;
    struct view_multi_cache tempcache;  /* temporary storage */

    tempcache = *current_cache;

    for (i = matched_cache; i > 0; i--)   {  /* go in reverse order */
        view_1_multi_cache[i] = view_1_multi_cache[i-1];
    }

    view_1_multi_cache[0] = tempcache;  /* copy matched cache to 1st pos'n */
}

/* make pixmap and set all members of view_1_multi_cache[0] */
void
create_multi_cache_member()
{
    int i;
    Flag matched_size;  /* true if pixmap, window size in struct = current values */

    struct view_multi_cache *create_cache; /* pointer to cache to be set */
    View *v;
```

```
v = views[1];

/* assign pixmaps to all multi_caches */
for(i = 0; i < MaxCache; i++)  {
   create_cache = &view_1_multi_cache[i];

   if((create_cache->pix_width != v->X_x2) &&
     (create_cache->pix_height != v->X_y1) &&
     (create_cache->x_width != s->width) &&
     (create_cache->x_height != s->height))  {

     /* set pixmap depth */
     int view_1_cache_depth = visualList[0].depth;

     if(view_1_cache_depth == 0) view_1_cache_depth = 8;
     if(view_1_cache_depth != 8) printf("pixmap depth not 8?!?\n");

     matched_size = False;

     if(create_cache->pixels)  {
        int free_mc_return;  /* returned value from free_mc_pixmap() */
        free_mc_return = free_multi_cache_pixmap(create_cache);
     }

     /* set pixmap width, depth */
     create_cache->pix_width = v->X_x2;
     create_cache->pix_height = v->X_y1;

     /* create the pixmap */
     create_cache->pixels = XCreatePixmap(s->display, s->win,
                         create_cache->pix_width,
                         create_cache->pix_height,
                         view_1_cache_depth);

     /* window size when created */
```

```
            create_cache->x_width = s->width;
            create_cache->x_height = s->height;

            matched_size = True;

         } /* end inner if statement */

         else  /* all the pixmap, window size values are set properly */
            matched_size = True;

      } /* end for statement */

      if(matched_size)  {
         printf("copying screen to multi_cache pixmap...\n");

         create_cache = &view_1_multi_cache[0];  /* create pixmap in 1st entry */

         /* coordinates of background map when pixmap stored */
         create_cache->back_origin_x = v->window_x1;
         create_cache->back_origin_y = v->window_y1;
         create_cache->back_corner_x = v->window_x2;
         create_cache->back_corner_y = v->window_y2;

         /* store contents of screen into created pixmap */

         XCopyArea(s->display, s->win, create_cache->pixels,
               s->gc, 0, 0,
               create_cache->pix_width,
               create_cache->pix_height,
               0, 0);

         create_cache->pixmap_assigned = True;
      }
   }
```

```
/* copy top member of cache array to screen -- all structure values already checked */
void
copy_multi_cache_to_screen()
{
  if(view_1_multi_cache[0].pixmap_assigned == True)
    XCopyArea(s->display, view_1_multi_cache[0].pixels,
          s->win, s->gc, 0, 0,
          view_1_multi_cache[0].pix_width,
          view_1_multi_cache[0].pix_height,
          0, 0);
}

/* initialize all members of multi_cache array */
void
multi_cache_initialize()
{
  int i;

  for (i = 0; i < MaxCache; i++) {
    view_1_multi_cache[i].pix_width = 0;
    view_1_multi_cache[i].pix_height = 0;
    view_1_multi_cache[i].x_width = 0;
    view_1_multi_cache[i].x_height = 0;
    view_1_multi_cache[i].back_origin_x = 0;
    view_1_multi_cache[i].back_origin_y = 0;
    view_1_multi_cache[i].back_corner_x = 0;
    view_1_multi_cache[i].back_corner_y = 0;
    view_1_multi_cache[i].pixmap_assigned = False;

  }
}

/* Free structure's pixmap member */
int
free_multi_cache_pixmap(free_pixels)
```

```
struct view_multi_cache *free_pixels;
{
   if(free_pixels->pixels) {
      XFreePixmap(s->display, free_pixels->pixels);
      free_pixels->pixmap_assigned = False;
      return 1;
   }

   else    return 0;
}

/* Free all used pixmaps on program termination */
void
free_all_multi_cache_pixmaps()
{
   int i;
   struct view_multi_cache *free_pixels;
   int cache_freed;

   for(i = 0; i < MaxCache; i++) {
      free_pixels = &view_1_multi_cache[i];
      cache_freed = free_multi_cache_pixmap(free_pixels);
   }
}
```

Changes to Tksim.c

```
/* the modifications to tksim.c */

end_segment()
{
        Segment *segment, *find_it();
        int segnum;

   /* E Davies -- 97/09/24 -- primlist counter variables
      Primlist *scrollprimptr;  -- used for counting # of primlists per segment --
```

```
    int primcount;          -- primcounter variable --
    static int totpcount = 0; -- total number of primitives --
    int primsize;           -- size of prims in each segment --
    static int cumprimsize = 0; -- cumulative prim size  --
  end of declaration modifications -- E Davies */

      if (!new_segment_open) punt("end seg when one's not open?");

      if (future_seg_visible)
        new_segment.views = 1 << (current_view_num - 1);

      segnum = new_segment.segnum;   /* this is silly */

/*    printf(" tksim: end segnum = %d\n",segnum);  */


      new_segment_open = FALSE;
      install_newsegment();

      segment = find_it(segnum);

/*    printf(" tksim.c: 2: end_segment:   Segnum = %d\n",segnum); */

      set_bbox(segment);          /* fill in the bbox pls */

/*    printf(" tksim.c: 3: end_segment:   Segnum = %d\n",segnum); */

      paint_segment(segment, FALSE);    /* and paint it pls */
      /** paint_sync(); **/

/*    printf(" tksim.c: end: end_segment:   Segnum = %d\n",segnum);  */

    /* E Davies -- 97/09/24 -- start of primlist counting code */
      /*  scrollprimptr = segment->prims;  -- assign address of head of prim list
            to scrollprimptr --
```

```
        primcount = 0, primsize = 0;
        while(scrollprimptr)          -- while primlist *next != NULL --
        { */
          /*  primsize += scrollprimptr->size;    -- add size of prims */
          /*  primcount++;                        -- primlist counter */
          /*  scrollprimptr = scrollprimptr->next; -- move scrollprimptr to next Primlist --
        } */

       /* printf("  primcount = %d, primsize = %d\n", primcount, primsize);
        cumprimsize += primsize;
        totpcount += primcount;
        printf("  total primnum = %d, total primsize = %d\n",
           totpcount, cumprimsize); */

    /* end of end_segment modification -- E Davies */
}

/* this function is the second last function in tksim.c */
segment_reports()  /* information on number of segments, segment size -- same
                      for primlists */
{
#ifdef old                  /* E Davies -- 97/09/30 -- function now works */
        Segment *s;
        Primlist *p;
        u_long totalprim = 0;  /* total memory used by primlists */
        u_long segcount  = 0;  /* total number of segments */
        u_long primcount = 0;  /* total number of primlists */

        s = seg_pri_list ;  /* initialize s to head of segment chain */

        while ( s ) { /* cycle through segment priority chain */
                int i = 0;       /* prim counter for each segment */
                int primsize = 0; /* size of single segment's prims */
                segcount++;
```

```
            p = s->prims;  /* set p to head of prim list for next s */

            while( p ) {  /* cycle through chain of primlists */
                    int curprimsize = 0; /* for size of current prim */
                    curprimsize = s->prims->size;
                    primsize += curprimsize;
                    i++;
                    primcount++;
                    /* prim_contents(p, curprimsize);  */
                        /* Function to check contents of primbufs */
                    p = p->next;
            }
            totalprim += primsize;
            printf("%6d, segnum = %4d,  primnum = %4d,  primsize = %8d\n",
                    s->segnum, segcount, i, primsize);

            s = s->next_pri; /* next segment in priority chain */
        }
        printf("%d total segments\n",       segcount);
        printf("%d total bytes in prims\n", totalprim);
        printf("%d total prims\n",          primcount);

#endif
}
```

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA
(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared e.g. Establishment Sponsoring a contractor's report, or tasking agency, are entered in Section 8).

Operational Research Division
Department of National Defence
Ottawa, Ontario K1A 0K2

2. SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title)

JANUS RTX IMPROVEMENTS

4. AUTHORS (last name, first name, middle initial)

Davies, Evan

5. DATE OF PUBLICATION (month Year of Publication of document)	6a. NO OF PAGES (total containing information. Include Annexes, Appendices, etc.)	6b. NO OF REFS (total cited in document)
December 1997	53	2

7. DESCRIPTIVE NOTES (the category of document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Research Note

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address).

DOR(J&L)

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written.)
3552-22242-10	---

10a. ORIGINATOR's document number (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DOR(J&L) Research Note RN 9716	---

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors: further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

This report documents the development of the pixmap cache, which speeds up reloading of terrain maps, thus reducing frustrating periods of inaction for the gamers. Several other features of the rtx code encountered during program investigation have also been documented to aid in future code revisions.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified . If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Janus